Embedded WPE WebKit
From Bring-Up to Maintenance

Adrian Perez de Castro

aperez@igalia.com

igalia

About Me

- CS Engineer, partner of Igalia.
- Systems person turned web engine developer.
- WebKit jack of all trades since ~2012.
- Current focus: platform layer, hardware bringup, release management.
- I like old computers, too!

About Igalia

- Specialized **Open Source consultancy**, founded in 2001
- **Fully remote**, HQ in **A Coruña** (Spain). **Flat structure**.
- **Top contributors** to all the main **Web Rendering Engines**
  - WebKit, Chromium, Gecko and Servo
- **Active contributor to other areas and OSS projects**
  - V8, SpiderMonkey, JSC, LLVM, Node.js, GStreamer, Mesa, Linux Kernel...
- **Members of several working groups**:
  - W3C, WHATWG, WPT, TC39, OpenJS, Test262, Khronos...

igalia
Open Source Consultancy

Agenda

1. WPE WebKit 101
2. Adding WPE to your device
3. Done! Now what?

WPE WebKit 101

WPE

101: What is WebKit

- Open Source **Web Engine**:
  - Ingests HTML/CSS/JavaScript/etc.
  - Produces rendered content.
- **Port**-able:
  - Port = Adaptation for a specific platform.
  - Runs in more places than one may imagine.
  - The WPE port focuses on Linux-based embedded devices.
- Multi-process architecture.

101: Why a Web Engine?

~~Software~~ The Web is eating the world

101: Why a Web Engine?

~~Software~~ The Web is eating the world

"There is ~~an app~~ a website for that."

101: Why a Web Engine?

~~Software~~ The Web is eating the world

"There is ~~an app~~ a website for that."

Embedded hardware is powerful enough ~~now~~ since ~2013.

101: Why a Web Engine?

~~Software~~ The Web is eating the world

"There is ~~an app~~ a website for that."

Embedded hardware is powerful enough ~~now~~ since ~2013.

Prototype and iterate ~~harder, better,~~ faster ~~stronger~~.

101: Why WPE WebKit?

Platformless WebKit port. **Designed** to be *embeddable*:

101: Why WPE WebKit?

Platformless WebKit port. **Designed** to be *embeddable*:

Extensible,

101: Why WPE WebKit?

Platformless WebKit port. **Designed** to be *embeddable*:

Extensible,

Adaptable,

101: Why WPE WebKit?

Platformless WebKit port. **Designed** to be *embeddable*:

Extensible,

Adaptable,

Minimal dependencies.

101: Why WPE WebKit?

Available *yesterday*:

101: Why WPE WebKit?

Available *yesterday*:

Build systems: Yocto, Buildroot, PTXdist;

101: Why WPE WebKit?

Available *yesterday*:

Build systems: Yocto, Buildroot, PTXdist;

Distributions: Debian, Ubuntu, Arch.

101: Why WPE WebKit?

Available *yesterday*:

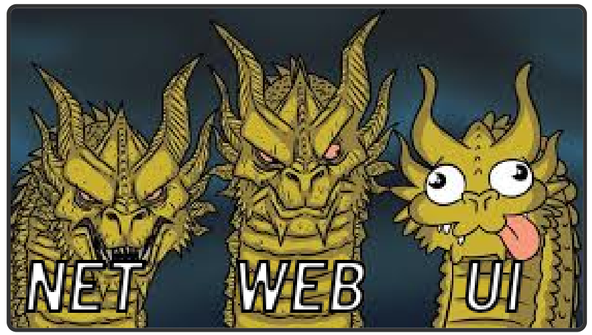
Build systems: Yocto, Buildroot, PTXdist;

Distributions: Debian, Ubuntu, Arch.

WPE has existed for 6+ years.

101: Multiprocess WebKit

- **UI Process**
  - Embeds a `WebKitWebView`.
  - This is your ~~browser~~ application.
- **WebProcess**:
  - Handles Web content.
  - Many of them, per-site isolation.
- **Network Process**



Adding WPE to Your Device

Act I: Bringup

Check List

Is my **hardware supported**?

Check List

Is my **hardware supported**?

Yes

It's your lucky day!

Check List

Is my **hardware supported**?

Yes

It's your lucky day!

No

There are some **requirements**.

Check List

Minimum	Desirable
OpenGL ES 2.0	64-bit CPU
Buffer sharing	EGL/GLES extensions

Check List

Minimum	Desirable
OpenGL ES 2.0	64-bit CPU
Buffer sharing	EGL/GLES extensions

`EGL_KHR_image_base`
`EGL_KHR_surfaceless_context`
`EGL_EXT_image_dma_buf_import`
`EGL_EXT_image_dma_buf_import_modifiers`
`EGL_MESA_image_dma_buf_export`

Pain Point: GPU Drivers 🖥

Pain Point: GPU Drivers 

Wayland/DMA-BUF support

Lack of OpenGL ES 3+

Vulkan-only drivers

Unusable driver

Pain Point: GPU Drivers 

Wayland/DMA-BUF support	Steadily improving
Lack of OpenGL ES 3+	GLESv2 as baseline
Vulkan-only drivers	Zink usable, seldom tested
Unusable driver	Mesa swrast/llvmpipe

Pain Point: GPU Drivers 

Wayland/DMA-BUF support	Steadily improving
Lack of OpenGL ES 3+	GLESv2 as baseline
Vulkan-only drivers	Zink usable, seldom tested
Unusable driver	Mesa swrast/llvmpipe

Prefer Open Source drivers 

Pain Point: “Funny” HW 

Pain Point: “Funny” HW 

Not 32-bpp

- `RGB565` incantation.
- DMA-BUF.
- Little testing.

Pain Point: “Funny” HW 

Not 32-bpp

- `RGB565` incantation.
- DMA-BUF.
- Little testing.

Rotated/Flipped

- Works on Wayland.
- Okay-ish on DRM/KMS.

Pain Point: “Funny” HW 

Not 32-bpp

- `RGB565` incantation.
- DMA-BUF.
- Little testing.

Rotated/Flipped

- Works on Wayland.
- Okay-ish on DRM/KMS.

Bare metal

- Exotic connections.
- Legacy: `fbdev` 😓.

Pain Point: “Funny” HW 

Not 32-bpp

- `RGB565` incantation.
- DMA-BUF.
- Little testing.

Rotated/Flipped

- Works on Wayland.
- Okay-ish on DRM/KMS.

Bare metal

- Exotic connections.
- Legacy: `fbdev` 😓.

Prefer common technologies 💫

Adding WPE to Your Device

Act II: Browser

Choosing a Browser

(In increasing order of complexity.)

Choosing a Browser

(In increasing order of complexity.)

Use **Cog**.

Choosing a Browser

(In increasing order of complexity.)

Use **Cog**.

Use **libcogcore**.

Choosing a Browser

(In increasing order of complexity.)

Use **Cog**.

Use **libcogcore**.

Roll your own.

Browser: libcogcore

```c
#include <cog/cog.h>

static const char *s_starturl = NULL;

static WebKitWebView* on_create_view(CogShell *shell, CogPlatform *platform) {
    WebKitWebViewBackend *view_backend = cog_platform_get_view_backend(platform, NULL, NULL);
    g_autoptr(WebKitWebView) web_view = g_object_new(WEBKIT_TYPE_WEB_VIEW,
        "settings", cog_shell_get_web_settings(shell), "web-context", cog_shell_get_web_context(shell),
        "backend", view_backend, NULL);
    cog_platform_init_web_view(platform, web_view);
    webkit_web_view_load_uri(web_view, s_starturl);
    return g_steal_pointer(&web_view);
}

int main(int argc, char *argv[]) {
    g_set_application_name("minicog");
    if (argc != 2 && argc != 3) g_error("Usage: %s [URL [platform]]\n", argv[0]);
    s_starturl = cog_uri_guess_from_user_input(argv[1], TRUE, NULL);

    cog_modules_add_directory(COG_MODULEDIR);
    g_autoptr(GApplication) app = g_application_new(NULL, G_APPLICATION_DEFAULT_FLAGS);
    g_autoptr(CogShell) shell = cog_shell_new("minicog", FALSE);
    g_autoptr(CogPlatform) platform = cog_platform_new(g_getenv("COG_PLATFORM"), NULL);
    cog_platform_setup(platform, shell, "", NULL);
    g_signal_connect(shell, "create-view", G_CALLBACK(on_create_view), platform);
    g_signal_connect_swapped(app, "shutdown", G_CALLBACK(cog_shell_shutdown), shell);
    g_signal_connect_swapped(app, "startup", G_CALLBACK(cog_shell_startup), shell);
    g_signal_connect(app, "activate", G_CALLBACK(g_application_hold), NULL);

    return g_application_run(app, 1, argv);
}
```


Browser: WPE Platform API

```c
#include <wpe/webkit.h>

int main(int argc, const char *argv[]) {
    g_autoptr(GMainLoop) loop = g_main_loop_new(NULL, FALSE);
    g_autoptr(WebKitWebView) view = webkit_web_view_new(NULL);

    webkit_web_view_load_uri(view,
        (argc > 1) ? argv[1] : "https://wpewebkit.org");
    g_main_loop_run(loop);

    return EXIT_SUCCESS;
}
```


Adding WPE to Your Device

Act III: Integration

Motivation

Make Web content *talk* to your hardware

Motivation

Make Web content *talk* to your hardware

Local web server!

Motivation

Make Web content *talk* to your hardware

Integration: URI Scheme Handler

Integration: URI Scheme Handler

```
static void
configure_web_context(WebKitWebContext *context) {
  webkit_web_context_register_uri_scheme(context,
    "echo",
    (WebKitURISchemeRequestCallback) handle_echo_scheme,
    NULL /* userdata */,
    NULL /* destroy_notify */);
}
```


Integration: URI Scheme Handler

```
static void
handle_echo_scheme(WebKitURISchemeRequest *r) {
  const char *uri = webkit_uri_scheme_request_get_uri(r);

  g_autoptr(GBytes) data = g_bytes_new(uri, strlen(uri));
  g_autoptr(GInputStream) stream =
        g_memory_input_stream_new_from_bytes(data);

  webkit_uri_scheme_request_finish(request, stream,
        g_bytes_get_size(data), "text/plain");
}
```


Integration: URI Scheme Handler

```
static void
handle_echo_scheme(WebKitURISchemeRequest *r) {
  const char *uri = webkit_uri_scheme_request_get_uri(r);

  g_autoptr(GBytes) da                          strlen(uri));
  g_autoptr(GI
        g_memo                                  s(data);

  webkit_uri_scheme_request_finish(request, stream,
        g_bytes_get_size(data), "text/plain");
}
```

Some fine print applies

⚠️ CORS Ahead! ⚠️

Integration:
User Script Messages

Integration: User Script Messages

```
static void
configure_content_manager(WebKitUserContentManager *mgr) {
  webkit_user_content_manager_register_script_message_handler_with_reply(
        mgr, "bluetooth", NULL);
  g_signal_connect(mgr,
        "script-message-with-reply-received::bluetooth",
        G_CALLBACK(handle_bluetooth), NULL);
}
```


Integration: User Script Messages

```
static gboolean
handle_bluetooth(WebKitUserContentManager *mgr, JSCValue *value,
      WebKitScriptMessageReply *reply) {
  g_autoptr(JSCValue) result = /* ... */;
  webkit_script_message_reply_return_value(reply, result);
}
```


Integration: User Script Messages

```
function PairBluetooth(btAddr) {
  const m = { kind: "pair", target: btAddr };
  return window.webkit.messageHandlers.bluetooth.postMessage(m);
}

// Elsewhere:
const device = await PairBluetooth("FD:FD:34:00:00:01");
console.log("Paired with:", device.name);
```


Integration: User Script Messages

```
function PairBluetooth(btAddr) {
  const m = { kind: "pair", target: btAddr };
  return window.webkit.messageHandl                th.postMessage(m);
}

// Elsewhere:
const device = awa                .Bluetooth("FD:FD:34:00:00:01");
console.log("Paired with:", device.name);
```

📨 **Async**

Involves message passing

Integration: JSC API

Integration: JSC API

```
static void native_print(JSCValue *value) {
  g_autofree char *value_string =
        jsc_value_to_string(value);
  printf("%s\n", value_string);
}
```

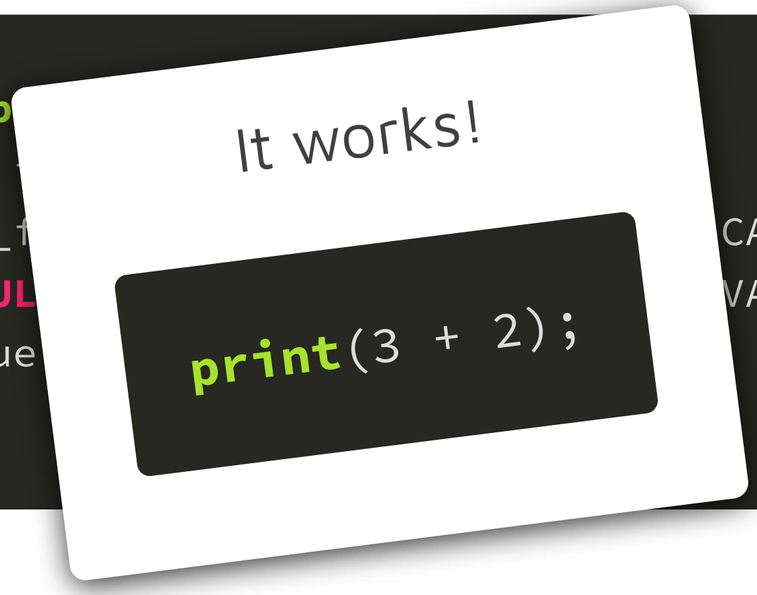

Integration: JSC API

```
static void register_print(JSCContext *context) {
  g_autoptr(JSCValue) func =
        jsc_value_new_function(context, "print", G_CALLBACK(native_print),
              NULL, NULL, G_TYPE_NONE, 1, JSC_TYPE_VALUE);
  jsc_context_set_value(context, "print", func);
}
```


Integration: JSC API

```
static void register_p
  g_autoptr(JSCValue)
        jsc_value_new_f
              NULL, NUL
  jsc_context_set_value
}
```

```
CALLBACK(native_print),
VALUE);
```



Integration: JSCContext

- Standalone program:
  - `jsc_context_new()`.
- Web process extension
  - In the Web Process:
    - `WebKitWebExtension::page-created`
      - `webkit_script_world_get_default()`
      - `WebKitScriptWorld::window-object-cleared`
        - `webkit_frame_get_js_context_for_script_world()`
  - In the IU Process (browser):
    - `webkit_web_context_set_web_process_extensions_directory()`

Integration: Resources

- Blog: **URI Scheme Handlers and Script Messages**.
- Reference: **WPE WebKit API**.
- Reference: **Web Process Extension API**.
- Reference: **JSC API**.

Done, now what?

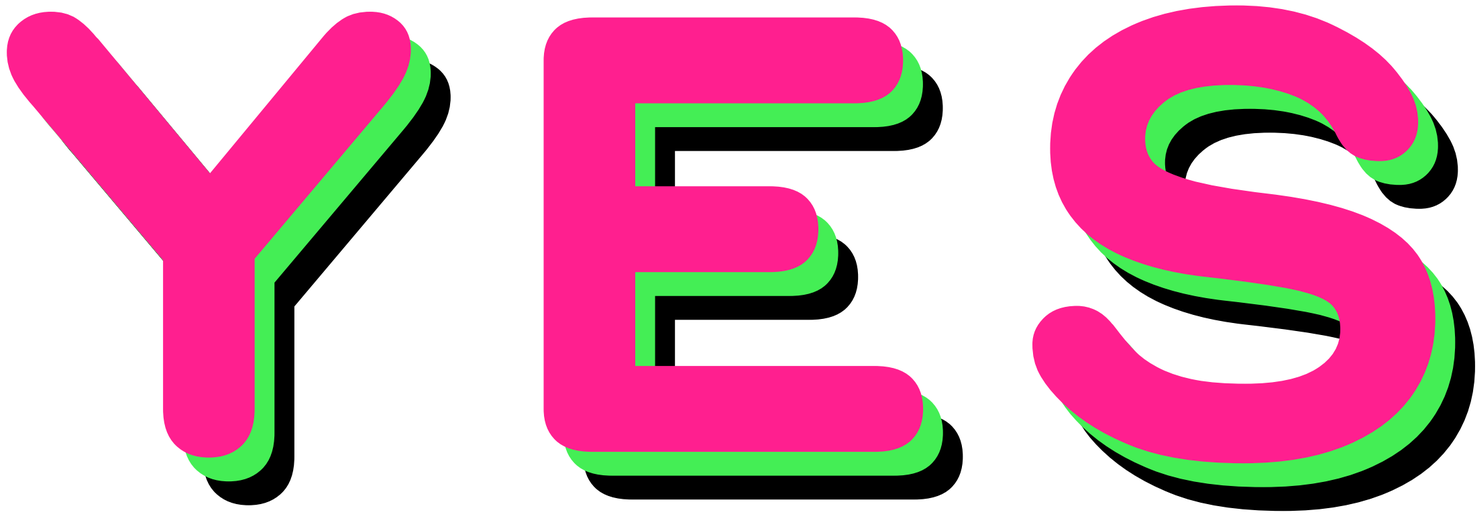
Now What? Updates

We are trying hard to make updating WPE WebKit hassle-free:

- Feature **releases** each March and September.
- Documented dependencies **policy**, **GCC requirements**.
- Recommended **practices for security updates**.
- Stable API + ABI. New features typically backwards-compatible.

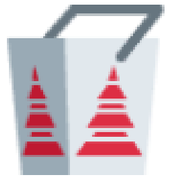
Should I ship updates?

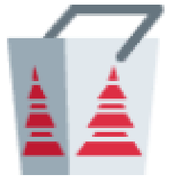
Should I ship updates?



Updates — Some tips

- Pick minor releases with security fixes. These rarely break.
- Consider using WebDriver for automated testing.
- Avoid patching WPE WebKit, follow upstream if possible.
- Parallel integration queue that uses development releases.
- Design your QA for testing updates, too!

Takeaways 

Takeaways 

- WPE *can* fit your needs.
- Hardware checklist.
- Prefer OSS drivers.
- Prefer common hardware.
- Do ship updates.
- Avoid rolling your browser.
- Reuse parts of another.
- Avoid local HTTP servers.
- Try easier integration.
- QA for WPE + Web apps.

Q&A

Bonus Track

- WPE WebKit can be used headless.
- The JSC API can be used standalone.
- Resource usage can be limited with `cgroups`
  - Memory limits.
  - CPU usage.
- Hardening:
  - Run as non-`root`.
  - Use containers/namespaces.
  - Built-in Bubblewrap sandbox.


igalia